PATENT ABSTRACTS OF JAPAN

(11)Publication number : **2004-086921**
(43)Date of publication of application : **18.03.2004**

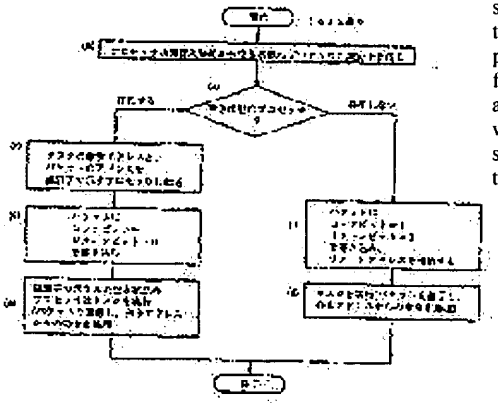
(51)Int.Cl. G06F 15/177
G06F 9/46

(21)Application number : **2003-363920**
(22)Date of filing : **23.10.2003**

(71)Applicant : **MATSUSHITA ELECTRIC IND CO LTD**
(72)Inventor : **TANAKA TETSUYA**
FUKUDA AKIRA
TANUMA HITOSHI

(30)Priority
Priority number : **07036836** Priority date : **24.02.1995** Priority country : **JP**

(54) MULTIPROCESSOR SYSTEM AND METHOD FOR EXECUTING TASK IN MULTIPROCESSOR SYSTEM THEREOF



(57)Abstract:
PROBLEM TO BE SOLVED: To provide a task executing method suitable to a task of fine granularity.
SOLUTION: This multiprocessor system comprises a step wherein whether or not there is a processor having a 'free state' among the processors 30-32 when a processor which is executing a task T1 among the processors 30-32 generates a new task T2, a step wherein when the processor having the 'free state' is detected, the task T2 begins to be executed by the processor by being assigned to the processor and the state of the processor is changed from the 'free state' to an 'execution state' to store a flag having a first value indicating that the execution of the task T1 is not interrupted. The system also includes a step wherein the execution of the task T1 is interrupted and the task T2 which is interrupted begins to be executed by the processor to store a flag having a second value indicating that the execution of the task T1 has been interrupted when the processor having a 'free state' is not detected.

LEGAL STATUS
[Date of request for examination] 23.10.2003
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-86921
(P2004-86921A)
(43)公開日　平成16年3月18日(2004.3.18)

(51)Int.Cl.[7]	F I			テーマコード（参考）
G06F 15/177	G06F	15/177	674A	5B045
G06F 9/46	G06F	9/46	360B	5B098

審査請求 有　請求項の数 1　OL　（全 21 頁）

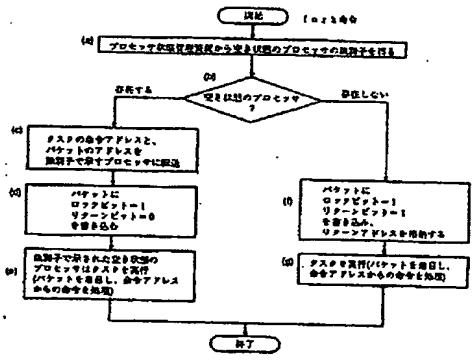
(21)出願番号　特願2003-363920(P2003-363920)
(22)出願日　平成15年10月23日(2003.10.23)
(62)分割の表示　特願平8-36964の分割
原出願日　平成8年2月23日(1996.2.23)
(31)優先権主張番号　特願平7-36836
(32)優先日　平成7年2月24日(1995.2.24)
(33)優先権主張国　日本国(JP)

(71)出願人　000005821
松下電器産業株式会社
大阪府門真市大字門真１００６番地
(74)代理人　100078282
弁理士　山本　秀策
(74)代理人　100062409
弁理士　安村　高明
(74)代理人　100107489
弁理士　大塩　竹志
(72)発明者　田中　哲也
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72)発明者　福田　晃
福岡県三潴郡城島町大字青木島１９９番地２

最終頁に続く

(54)【発明の名称】マルチプロセッサシステムおよびマルチプロセッサシステムにおいてタスクを実行する方法

(57)【要約】
【課題】　細粒度のタスクに適したタスクの実行方法を提供する。
【解決手段】　プロセッサ３０～３２のうちタスクＴ１を実行中のプロセッサが新たなタスクＴ２を生成した場合において、プロセッサ３０～３２のうち「空き状態」を有するプロセッサがあるか否かを検出するステップと、「空き状態」を有するプロセッサが検出された場合には、タスクＴ２をそのプロセッサに割り当てることにより、そのプロセッサによるタスクＴ２の実行を開始し、そのプロセッサの状態を「空き状態」から「実行状態」に変更し、タスクＴ１の実行が中断されていないことを示す第１の値を有するフラグを格納するステップと、「空き状態」を有するプロセッサが検出されない場合には、タスクＴ１の実行を中断し、中断したプロセッサによるタスクＴ２の実行を開始し、タスクＴ１の実行が中断されたことを示す第２の値を有するフラグを格納するステップとを包含する方法。
【選択図】　図７



【特許請求の範囲】
【請求項１】
「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいてタスクを実行する方法であって、
該複数のプロセッサのうち第１タスクを実行中の第１プロセッサが新たな第２タスクを生成した場合において、該第１プロセッサが、該第２タスクの実行を依頼するために、該複数のプロセッサのうち「空き状態」を有する第２プロセッサがあるか否かを検出するステップと、
該検出の結果、「空き状態」を有する第２プロセッサが検出された場合には、該第２タスクを該第２プロセッサに割り当てることにより、該第２プロセッサによる該第２タスクの実行を開始し、該第２プロセッサの状態を「空き状態」から「実行状態」に変更し、該第１タスクの実行が中断されていないことを示す第１の値を有するフラグを格納するステップと、
該検出の結果、「空き状態」を有する第２プロセッサが検出されない場合には、該第１プロセッサによる該第１タスクの実行を中断し、該第１プロセッサによる該第２タスクの実行を開始し、該第１タスクの実行が中断されたことを示す第２の値を有するフラグを格納するステップと
を包含する方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、複数のタスクを並列に実行する複数のプロセッサを含むマルチプロセッサシステムおよびそのマルチプロセッサシステムにおいてタスクを実行する方法に関する。
【背景技術】
【０００２】
近年、マルチプロセッサシステムは汎用計算機の並列処理による高性能化のアプローチの一つとして注目されている。マルチプロセッサシステムにおいては複数のプロセッサを一つのバスに接続し、主記憶装置を共有する共有メモリ型のマルチプロセッサシステムが主に採用されている。
【０００３】
このようなマルチプロセッサシステムは通常、複数のプロセッサチップをプリント基板上に実装するため、各プロセッサの処理速度に対し、プロセッサ間のバスを用いる通信や同期の処理速度は遅い。そのため、処理単位であるタスクの処理時間がプロセッサ間の通信や同期の時間に対し十分大きい場合に用いられる。この場合のタスクの大きさは中粒度～粗粒度と呼ばれ実行命令数で数１０００命令程度以上とされている。このように、処理単位を大きくする（粒度を粗くする）ことでタスクの実行時間に対して相対的にプロセッサ通信や同期の時間を小さくしている。
【０００４】
さらに、近年半導体の集積化技術は急速に発展している。そのため、チップ内に多くの機能ユニットやメモリを搭載することができるようになってきている。マルチプロセッサシステムにおいても今後複数のプロセッサをワンチップに搭載することが可能になると思われる。その場合、プロセッサが接続されるバスもチップ内に入ることになりプロセッサ間の通信や同期の高速化はそこで実行するタスクの粒度の選択肢を広げる。即ち、タスクの大きさが細粒度、命令数で数１０～数１００命令程度の並列処理が可能になりつつある。今後、このような細粒度のタスクを並列処理することが主流になると予想される。近年注目されているオブジェクト指向プログラミングや関数型言語を用いたプログラミングは、いずれも「細粒度のタスクを並列処理する」ことに合致したものであるからである。
【０００５】
一方、マルチプロセッサシステムでは、複数のタスクを物理的に限られたプロセッサ数に割り当てることになるため、タスクの実行順序を決定し、どのプロセッサに対しどのタ

スクを割り当てるかを適切に選択することが行われる。この処理を動的に行うため、まず実行待ちタスクを一次記憶などのタスク管理装置に格納しておき、次に空きプロセッサを検出し、空きプロセッサがある場合は、実行待ちタスクの中から実行すべきタスクを選択し、選択したタスクを空きプロセッサに割り当てることが行われる。このときのタスク選択は仕事全体の実行時間を最小にするなどの目的で行われる。こういったタスクの実行順序を決定し、タスクをどのプロセッサに割り当てるかを決定する処理をスケジューリングといい、決定方法の異なるさまざまなアルゴリズムがある。また、タスク生成によって実行すべきタスクが生じた場合、タスク管理装置に実行待ちタスクとして登録する処理もある。

【０００６】

図１２にマルチプロセッサシステムにおける、従来のプロセッサ割当方法の動作説明図を示す。図１２において、プロセッサ２はタスクを生成し、実行待ちのタスクとしてタスク管理装置にタスク４を登録している。プロセッサ０はプロセッサ１が「空き状態」であることを検出すると、タスク管理装置の実行待ちのタスクをスケジューリングアルゴリズムにしたがって一つを選択し、選択されたタスクはプロセッサ０によりプロセッサ１に割り当てられる。このとき、プロセッサ０はスケジューリングの処理を、プロセッサ２はタスク登録の処理をそれぞれ行っている。

【０００７】

これは、例えば特開昭６３－２０８９４８号公報に示すように空きプロセッサ（図１２ではプロセッサ１）がタスクレディーキュー（図１２ではタスク管理装置）の監視を行い、実行待ちのタスクを自動的に取り出し処理する場合でも、「空き状態」のプロセッサがスケジューリングの処理を行っている。

【０００８】

また、例えば特開昭６２－１９０５４８号公報に示されるように、タスクを依頼した依頼プロセッサが、依頼された被依頼プロセッサでのタスクの状態を監視しておき、被依頼プロセッサがタスクの終了を検出した場合、空きプロセッサとなった被依頼プロセッサにほかのタスクを適切に選択し割り当てる方法がある。この方法においては、依頼プロセッサが被依頼プロセッサの状態を監視する処理を行っている。

【０００９】

前記したスケジューリング処理やタスクの登録処理、もしくは被依頼プロセッサを監視する処理はそれぞれ内容は異なるもののタスクをプロセッサに割り当て実行するまでのオーバヘッド即ちタスク処理に付随するオーバヘッドと考えることができる。図１３はタスクの処理時間と前記したオーバヘッドの処理時間のタイムチャートを示している。図１３に示すようにタスクの粒度が中～粗粒度の場合はタスクの処理時間に対してオーバヘッドの処理時間が相対的に小さいため、オーバヘッドの処理時間を無視できるレベルにある。

【発明の開示】

【発明が解決しようとする課題】

【００１０】

しかしながら、上記のようなタスク処理に付随するオーバヘッドを持つマルチプロセッサシステムにおいて、プロセッサ間の通信や同期を高速化することで細粒度の並列処理を行う場合は、タスクの処理時間に対して相対的にオーバヘッドの処理時間が大きくなる。

【００１１】

図１４は細粒度の場合のタスクの処理時間とオーバヘッドの処理時間のタイムチャートを示している。図１４に示すようにオーバヘッドの処理時間はタスクの処理時間に比べて相対的に大きくなり、オーバヘッドの処理時間が無視できず仕事全体としての処理時間が大きくなるという問題を有する。

【００１２】

本発明は上記問題点に鑑み、細粒度の並列処理をプロセッサ間の通信や同期が高速なマルチプロセッサにおいて、タスク管理やスケジューリング、タスク状態の監視を行わないことで、前記したオーバヘッドをなくし、その代わりのプロセッサに対する動的なタスク

割当を一元的、単純かつ高速に行う方法を提供することにある。

【課題を解決するための手段】

【００１３】

本発明の方法は、「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいてタスクを実行する方法であって、該複数のプロセッサのうち第１タスクを実行中の第１プロセッサが新たな第２タスクを生成した場合において、該複数のプロセッサのうち「空き状態」を有する第２プロセッサがあるか否かを検出するステップと、「空き状態」を有する第２プロセッサが検出された場合には、該第２タスクを該第２プロセッサに割り当てることにより、該第２プロセッサによる該第２タスクの実行を開始し、該第２プロセッサの状態を「空き状態」から「実行状態」に変更し、該第１タスクの実行が中断されていないことを示す第１の値を有するフラグを格納するステップと、「空き状態」を有する第２プロセッサが検出されない場合には、該第１プロセッサによる該第１タスクの実行を中断し、該第１プロセッサによる該第２タスクの実行を開始し、該第１タスクの実行が中断されたことを示す第２の値を有するフラグを格納するステップとを包含しており、これにより上記目的が達成される。

【００１４】

前記方法は、前記第２タスクの実行が終了した後、前記フラグが前記第１の値と前記第２の値のうちのいずれを有するかを判定するステップと、前記フラグが前記第１の値を有すると判定された場合には、前記第２プロセッサの状態を「実行状態」から「空き状態」に変更するステップと、前記フラグが前記第２の値を有すると判定された場合には、前記第１タスクの実行が中断されたところから前記第１プロセッサによる前記第１タスクの実行を再開するステップとをさらに包含してもよい。

【００１５】

前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「空き状態」を有する第２プロセッサの検出は、該識別子を用いて行われてもよい。

【００１６】

前記複数のプロセッサのそれぞれは、タスクを割り当てる優先順位を決定する優先度を有しており、前記第２プロセッサへの前記第２タスクの割り当ては、該優先度に基づいて行われてもよい。

【００１７】

本発明の他の方法は、「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいて、「停止状態」と「第１実行状態」と「第２実行状態」とを有するタスクを実行する方法であって、該複数のプロセッサのうち第１タスクを実行中の第１プロセッサが新たな第２タスクを生成した場合において、該複数のプロセッサのうち「空き状態」を有する第２プロセッサがあるか否かを検出するステップと、「空き状態」を有する第２プロセッサが検出された場合には、該第２タスクを該第２プロセッサに割り当てることにより、該第２プロセッサによる該第２タスクの実行を開始し、該第２プロセッサの状態を「空き状態」から「実行状態」に変更し、該第２タスクの状態を「停止状態」から「第１実行状態」に変更するステップと、「空き状態」を有する第２プロセッサが検出されない場合には、該第１プロセッサによる該第１タスクの実行を中断し、該第１プロセッサによる該第２タスクの実行を開始し、該第２タスクの状態を「停止状態」から「第２実行状態」に変更するステップとを包含しており、これにより上記目的が達成される。

【００１８】

前記方法は、前記第２タスクの実行が終了した後、前記第２タスクの状態を判定するステップと、前記第２タスクが「第１実行状態」を有すると判定された場合には、前記第２プロセッサの状態を「実行状態」から「空き状態」に変更し、前記第２タスクの状態を「第１実行状態」から「停止状態」に変更するステップと、前記第２タスクが「第２実行状態」を有すると判定された場合には、前記第２タスクの状態を「第２実行状態」から「停

止状態」に変更するステップとをさらに包含してもよい。

【００１９】

前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「空き状態」を有する第２プロセッサの検出は、該識別子を用いて行われてもよい。

【００２０】

前記複数のプロセッサのそれぞれは、タスクを割り当てる優先順位を決定する優先度を有しており、前記第２プロセッサへの前記第２タスクの割り当ては、該優先度に基づいて行われてもよい。

【００２１】

本発明のマルチプロセッサシステムは、複数のタスクを並列に実行する複数のプロセッサと、該複数のプロセッサの状態を管理し、該複数のプロセッサのそれぞれからの問い合わせに応じて「空き状態」のプロセッサの識別子を返す状態管理手段とを備えており、該複数のプロセッサのそれぞれは、新たなタスクが発生した時点で、該状態管理手段に対して「空き状態」のプロセッサがあるか否かを問い合わる。これにより上記目的が達成される。

【００２２】

前記状態管理手段は、該プロセッサからの問い合わせに応答して、現在の状態を次の状態に遷移させる手段と、該次の状態に基づいて該問い合わせに対する応答を出力する手段とを備えていてもよい。

【００２３】

前記マルチプロセッサシステムは、該複数のプロセッサのそれぞれについて、命令キャッシュメモリとデータキャッシュメモリとをさらに備えていてもよい。

【００２４】

前記マルチプロセッサシステムは、前記複数のプロセッサ間で命令アドレスおよびパケットアドレスを転送するためのネットワークをさらに備えていてもよい。

【００２５】

該複数のタスクのそれぞれは、細粒度であってもよい。

【発明の効果】

【００２６】

本発明によれば、あるプロセッサで新たなタスクを生成したときにそのタスクの実行を他あるいは自プロセッサによりただちに開始することができる。このことは、タスクを保持しておく機構やタスクの実行順序をスケジューリングする機構を不要にする。また、実行待ちのタスクを選択し、その選択されたタスクを「空き状態」のプロセッサに割り当てる処理も不要となる。

【００２７】

その結果、タスクの処理時間に比較してプロセッサ割り当てに要する時間が少なくてすむ。これにより、マルチプロセッサシステムにおいて、粒度の細かい並列処理の高速化を図ることができる。

【発明を実施するための最良の形態】

【００２８】

以下、図面を参照しながら、本発明の実施の形態を説明する。

【００２９】

図１は、本発明のマルチプロセッサシステム１の構成を示す。マルチプロセッサシステム１は、集積回路上にインプリメントされる。マルチプロセッサシステム１は、バスを介して主記憶装置２に接続される。

【００３０】

マルチプロセッサシステム１は、要素プロセッサユニット１０～１２を含む。要素プロセッサユニット１０～１２のそれぞれは、同一の構成を有している。マルチプロセッサシステム１に含まれる要素プロセッサユニットの数は、３に限定されるわけではない。マル

チプロセッサシステム１は、任意の個数の要素プロセッサユニットを含み得る。

【００３１】

要素プロセッサユニット１０～１２は、それぞれ、プロセッサ３０～３２と命令キャッシュ（ＩＣ）３３～３５とデータキャッシュ（ＤＣ）３６～３８とを有している。命令キャッシュ（ＩＣ）は、命令を格納するためのキャッシュメモリであり、読み出し専用である。データキャッシュ（ＤＣ）は、データを格納するためのキャッシュメモリであり、読み出しと書き込みができる。

【００３２】

共有キャッシュ２０は、要素プロセッサユニット１０～１２によって共有されている。命令セットやデータセットは、通常、主記憶装置２に格納されている。データセットは、必要に応じてバスインタフェース２３を介して共有キャッシュ２０にロードされる。共有キャッシュ２０は、主記憶装置２と比較して非常に高速に動作することが好ましい。データキャッシュ（ＤＣ）と共有キャッシュ２０とは、アドレスに応じて使い分けられる。例えば、アドレスが０ｘ００００００００～０ｘ７ｆｆｆｆｆｆｆの範囲内である場合には、データキャッシュ（ＤＣ）がアクセスされ、アドレスが０ｘ８０００００００～０ｘｆｆｆｆｆｆｆｆの範囲内である場合には、共有キャッシュ２０がアクセスされる。

【００３３】

要素プロセッサユニット１０～１２は、ネットワーク２１を介して相互に接続される。ネットワーク２１は、要素プロセッサユニット１０～１２の相互間で命令アドレスやパケットアドレスを転送するために使用される。ネットワーク２１は、例えば、３×３のクロスバースイッチを用いて実現することができる。

【００３４】

プロセッサ状態管理装置２２は、プロセッサ３０～３２の状態を管理する。プロセッサ３０～３２のそれぞれは、「実行状態」および「空き状態」のいずれか一方の状態を有する。

【００３５】

プロセッサ３０～３２のそれぞれには固定された優先度が予め割り当てられている。ここでは、プロセッサ３０～３３は、この順番に高い優先度を有していると仮定する。優先度は、複数のプロセッサがプロセッサ状態管理装置２２を同時にアクセスする場合において、その複数のプロセッサのうちのどのプロセッサにプロセッサ状態管理装置２２に優先的にアクセスすることを許すかを決定するために使用される。

【００３６】

プロセッサ３０～３２のそれぞれは、プロセッサ３０～３２を互いに識別するための識別子（ＩＤ）を有している。典型的には、識別子（ＩＤ）は、番号によって表現される。

【００３７】

プロセッサ３０～３２のそれぞれは、それの内部にパケットのアドレスを保持する。パケットのアドレスは、例えば、プロセッサ３０～３２の内部のレジスタ（図示せず）に保持される。これにより、プロセッサ３０～３２は、パケットを参照することができる。パケットの詳細は、図６を参照して後述される。

【００３８】

マルチプロセッサシステム１は、複数のタスクを並列に実行する機能を有する。例えば、プロセッサ３０がタスクＴ１を実行しているのと並行して、プロセッサ３１はタスクＴ２を実行することができる。

【００３９】

本明細書では、「タスク」とは、命令セットとデータセットとの組であると定義する。命令セットとデータセットとは、いずれも主記憶装置２に格納される。プロセッサ３０～３２のそれぞれは、命令セットから命令を逐次読み出し、読み出された命令を解釈実行する。データセットは、プロセッサ３０～３２が命令セットから読み出された命令を解釈実行する際、必要に応じて参照される。また、後述されるパケットは、データセットの少なくとも一部である。

【００４０】
図２は、タスクの概念を模式的に示す。この例では、タスク１は、命令セット１とデータセット１の組によって定義され、タスク２は、命令セット１とデータセット２の組によって定義され、タスク３は、命令セット２とデータセット３の組によって定義される。命令セット１～２とデータセット１～３は、それぞれ、主記憶装置２に格納されている。
【００４１】
図３は、プロセッサ３０～３２の状態を管理するプロセッサ状態管理装置２２の構成例を示す。プロセッサ状態管理装置２２は、入力（ＲＥＱ０～ＲＥＱ２、ＲＥＳＥＴ０～ＲＥＳＥＴ２）に応答して出力（ＩＤ０～ＩＤ２、ＮＭＰ０～ＮＭＰ２）を提供する組み合わせ回路を含んでいる。その組み合わせ回路は、現在の状態（Ｓ）と入力（ＲＥＱ０～ＲＥＱ２、ＲＥＳＥＴ０～ＲＥＳＥＴ２）とに応じて次の状態（ｎｅｘｔＳ）を決定し、次の状態に対応する出力（ＩＤ０～ＩＤ２、ＮＭＰ０～ＮＭＰ２）を提供する。現在の状態（Ｓ）から次の状態（ｎｅｘｔＳ）への遷移は、例えば、表１に示される状態遷移表に従って決定される。
【００４２】

【表１】

S	REQ	RESET	nextS	ID0	NMP0	ID1	NMP1	ID2	NMP2
001	001	000	011	01	0	--	-	--	-
010	010	000	011	--	-	00	0	--	-
100	100	000	101	--	-	--	-	00	0
011	001	000	111	10	0	--	-	--	-
011	010	000	111	--	-	10	0	--	-
011	011	000	111	10	0	--	1	--	-
101	001	000	111	01	0	--	-	--	-
101	100	000	111	--	-	--	-	01	0
101	101	000	111	01	0	--	-	--	1
110	010	000	111	--	-	00	0	--	-
110	100	000	111	--	-	--	-	00	0
110	110	000	111	--	-	00	0	--	1
111	001	000	111	--	1	--	-	--	-
111	010	000	111	--	-	--	1	--	-
111	100	000	111	--	-	--	-	--	1
111	011	000	111	--	1	--	1	--	-
111	101	000	111	--	1	--	-	--	1
111	110	000	111	--	-	--	1	--	1
111	111	000	111	--	1	--	1	--	1
011	000	001	010	--	-	--	-	--	-
011	000	010	001	--	-	--	-	--	-
011	001	010	101	10	0	--	-	--	-
011	010	001	110	--	-	10	0	--	-
101	000	001	100	--	-	--	-	--	-
101	000	100	001	--	-	--	-	--	-
101	001	100	011	01	0	--	-	--	-
101	100	001	110	--	-	--	-	01	0
110	000	010	100	--	-	--	-	--	-
110	000	100	010	--	-	--	-	--	-
110	010	100	011	--	-	00	0	--	-
110	100	010	101	--	-	--	-	00	0
111	000	001	110	--	-	--	-	--	-
111	000	010	101	--	-	--	-	--	-
111	000	100	011	--	-	--	-	--	-
111	000	011	100	--	-	--	-	--	-
111	000	101	010	--	-	--	-	--	-
111	000	110	001	--	-	--	-	--	-
111	001	010	101	--	1	--	-	--	-
111	001	100	011	--	1	--	-	--	-
111	001	110	001	--	1	--	-	--	-
111	010	001	110	--	-	--	1	--	-
111	010	100	011	--	-	--	1	--	-
111	010	101	010	--	-	--	1	--	-
111	100	001	110	--	-	--	-	--	1
111	100	010	101	--	-	--	-	--	1
111	100	011	100	--	-	--	-	--	1
111	011	100	011	--	1	--	1	--	-
111	101	010	101	--	1	--	-	--	1
111	110	001	110	--	-	--	1	--	1

図３において、Ｓは現在の状態、ＮｅｘｔＳは次の状態を示す。これらの状態は、プロセッサ３０～３２の状態を示す。例えば、Ｓ＝００１は、プロセッサ３０の状態が「実行状態」であり、プロセッサ３１とプロセッサ３２の状態が「空き状態」であることを示している。ＮｅｘｔＳについても同様である。

【００４３】

図３において、ＲＥＱ０～ＲＥＱ２は、プロセッサ３０～３２からプロセッサ状態管理装置２２に入力されるリクエストを表す。これらのリクエストは、「空き状態」のプロセッサの識別子を得ることをプロセッサ状態管理装置２２に依頼するものである。表１では、ＲＥＱ０～ＲＥＱ２をまとめてＲＥＱと表記している。例えば、ＲＥＱ＝１０１は、ＲＥＱ０が１（アサート）であり、ＲＥＱ１が０（ネゲート）であり、ＲＥＱ２が１（アサート）であることを示している。

【００４４】
図３において、ＲＥＳＥＴ０～ＲＥＳＥＴ２は、プロセッサ３０～３２からプロセッサ状態管理装置２２に入力されるリセットを表す。これらのリセットは、プロセッサ状態管理装置２２内に保持されているプロセッサ３０～３２の状態を「実行状態」から「空き状態」に変更することをプロセッサ状態管理装置２２に依頼するものである。表１では、ＲＥＳＥＴ０～ＲＥＳＥＴ２をまとめてＲＥＳＥＴと表記している。例えば、ＲＥＳＥＴ＝０１０は、ＲＥＳＥＴ０が０（ネゲート）であり、ＲＥＳＥＴ１が１（アサート）であり、ＲＥＳＥＴ２が０（ネゲート）であることを示している。
【００４５】
図３において、ＩＤ０～ＩＤ２は、プロセッサ３０～３２からのリクエストに対して「空き状態」のプロセッサの識別子を通知する信号を表す。これらの信号は、プロセッサ３０～３２からのリクエストに応答してプロセッサ状態管理装置２２から出力される。ＩＤ０～ＩＤ２の値の意味は、以下のとおりである。
【００４６】
００：プロセッサ３０が「空き状態」である。
【００４７】
０１：プロセッサ３１が「空き状態」である。
【００４８】
１０：プロセッサ３２が「空き状態」である。
【００４９】
図３において、ＮＭＰ０～ＮＭＰ２は、プロセッサ３０～３２からのリクエストに対して「空き状態のプロセッサが存在しない」旨を通知する信号を表す。これらの信号は、プロセッサ３０～３２からのリクエストに応答してプロセッサ状態管理装置２２から出力される。ＮＭＰ０～ＮＭＰ２の値の意味は、以下のとおりである。
【００５０】
０：「空き状態」のプロセッサが存在する。「空き状態」のプロセッサの識別子は、ＩＤ０～ＩＤ０２の値によって示される。
【００５１】
１：「空き状態」のプロセッサが存在しない。この場合、ＩＤ０～ＩＤ２の値は、ｄｏｎ’ｔ　ｃａｒｅである。
【００５２】
以下、図４と図５とを参照して、プロセッサ状態管理装置２２の機能および動作を説明する。プロセッサ状態管理装置２２は、マルチプロセッサシステムに含まれるすべてのプロセッサの状態を管理する。具体的には、プロセッサ状態管理装置２２は、プロセッサの識別子とプロセッサの状態とを一対にしてプロセッサ状態管理装置２２内に保持する。プロセッサの識別子は、複数のプロセッサを互いに識別するために使用される。典型的には、プロセッサの識別子は整数で表現される。プロセッサの状態は、「実行状態」か「空き状態」かのいずれかである。
【００５３】
プロセッサ状態管理装置２２は、あるプロセッサからのリクエストに応答して、「空き状態」のプロセッサが存在するか否かを判定する。「空き状態」のプロセッサが存在した場合には、プロセッサ状態管理装置２２は、その「空き状態」のプロセッサの識別子をそのリクエストを発したプロセッサに返す。「空き状態」のプロセッサが存在しなかった場合には、プロセッサ状態管理装置２２は、「空き状態のプロセッサが存在しない」旨のメッセージをそのリクエストを発したプロセッサに返す。
【００５４】
「空き状態」のプロセッサが複数個存在する場合には、プロセッサ状態管理装置２２は、「空き状態」の複数のプロセッサのうち優先度の最も高いプロセッサの識別子をそのリクエストを発したプロセッサに返す。また、複数のプロセッサからのリクエストが同時にプロセッサ状態管理装置２２に到達した場合には、そのリクエストを発した複数のプロセ

ッサのうち優先度の高いものから順に上述した処理が行われる。

【００５５】

図４（ａ）および（ｂ）は、プロセッサ状態管理装置２２の動作の一例を示す。プロセッサ状態管理装置２２は、４つのプロセッサ０～３の状態を管理している。図４（ａ）に示す例では、プロセッサ０とプロセッサ１の状態は「実行状態」であり、プロセッサ２とプロセッサ３の状態は「空き状態」である。プロセッサ０からのリクエストとプロセッサ１からのリクエストがプロセッサ状態管理装置２２に入力される。

【００５６】

プロセッサ状態管理装置２２は、プロセッサ０からのリクエストに応答して、「空き状態」のプロセッサ２の識別子をプロセッサ０に返し、プロセッサ１からのリクエストに応答して、「空き状態」のプロセッサ３の識別子をプロセッサ１に返す（図４（ｂ）参照）。「空き状態」のプロセッサの識別子は、プロセッサの優先度に従って返される。また、プロセッサ状態管理装置２２は、プロセッサ状態管理装置２２内に保持されているプロセッサ２の状態を「空き状態」から「実行状態」に変更し、プロセッサ３の状態を「空き状態」から「実行状態」に変更する。

【００５７】

図５（ａ）および（ｂ）は、プロセッサ状態管理装置２２の動作の他の一例を示す。プロセッサ状態管理装置２２は、４つのプロセッサ０～３の状態を管理している。図５（ａ）に示す例では、プロセッサ０とプロセッサ１とプロセッサ２の状態は「実行状態」であり、プロセッサ３の状態は「空き状態」である。プロセッサ０からのリクエストとプロセッサ１からのリクエストがプロセッサ状態管理装置２２に入力される。

【００５８】

プロセッサ状態管理装置２２は、プロセッサ０からのリクエストに応答して、「空き状態」のプロセッサ３の識別子をプロセッサ０に返し、プロセッサ１からのリクエストに応答して、「空き状態のプロセッサが存在しない」旨のメッセージをプロセッサ１に返す（図５（ｂ）参照）。「空き状態のプロセッサが存在しない」旨のメッセージは、例えば、プロセッサ状態管理装置２２から出力されるリターンコードの値によって表される。「空き状態」のプロセッサの識別子は、プロセッサの優先度に従って返される。また、プロセッサ状態管理装置２２は、プロセッサ状態管理装置２２内に保持されているプロセッサ３の状態を「空き状態」から「実行状態」に変更する。

【００５９】

図４と図５に示される例では、プロセッサ状態管理装置２２によって管理されるプロセッサの数は４である。しかし、これは、説明の便宜上のためであり、本発明が４つのプロセッサを有するマルチプロセッサシステムに限定されるわけではない。本発明は、任意の数のプロセッサを含むマルチプロセッサシステムに適用され得る。

【００６０】

図６は、パケット５０の構成を示す。パケット５０は、ロックビットを格納するロックビット領域５１と、リターンビットを格納するためのリターンビット領域５２と、リターンアドレスを格納するためのリターンアドレス領域５３と、引数を格納するための引数領域５４と、戻り値を格納するための戻り値領域５５とを有している。パケット５０は、タスク毎に共有メモリ２０上に確保され、タスクに所有される。これ以降、「タスクに所有されたパケット」を単に「タスクのパケット」と呼ぶ。パケット５０は、タスク間のデータの受け渡しやタスクの情報を保持するために使用される。

【００６１】

パケット５０のロックビット領域５１には、ロックビットが格納される。ロックビットは、パケット５０を所有するタスクが実行中である間、他のタスクからその実行中のタスクへのアクセスを禁止するか否かを示す。ロックビットが”１”であることは、アクセスを禁止していることを示す。ロックビットが”０”であることは、アクセスを禁止していないことを示す。

【００６２】

パケット５０のリターンビット領域５２には、リターンビットが格納される。リターンビットは、パケット５０を所有するタスクを実行する前に、他のタスクを中断したか否かを示す。リターンビットが”０”であることは、「パケット５０を所有するタスクを実行する前に、他のタスクを中断していない」ことを示す。これは、「空き状態」のプロセッサにパケット５０を所有するタスクが割り当てられた場合に相当する。リターンビットが”１”であることは、「パケット５０を所有するタスクを実行する前に、他のタスクを中断した」ことを示す。これは、「空き状態」のプロセッサが存在しなかったため、タスクを実行中のプロセッサがそのタスクの実行を中断して、パケット５０を所有する別のタスクを実行する
場合に相当する。

【００６３】

パケット５０のリターンアドレス領域５３には、リターンアドレスが格納される。リターンアドレスは、リターンビットが”１”である場合にのみ参照される。リターンアドレスは、中断されたタスクへの戻りアドレスを示す。

【００６４】

パケット５０の引数領域５４には、パケット５０を所有するタスクへの引数が格納される。

【００６５】

パケット５０の戻り値領域５５には、パケット５０を所有するタスクの実行結果である戻り値が格納される。

【００６６】

図７は、プロセッサ３０～３２がｆｏｒｋ命令を解釈実行する手順を示す。プロセッサ３０～３２は、主記憶装置２に格納されている命令セットから命令を読み出す。読み出された命令がｆｏｒｋ命令である場合には、プロセッサ３０～３２は、図７に示す処理を実行する。

【００６７】

以下、図７を参照して、プロセッサ３０がｆｏｒｋ命令を解釈実行する手順をステップごとに詳細に説明する。他のプロセッサ３１および３２がｆｏｒｋ命令を解釈実行する場合も同様である。なお、ｆｏｒｋ命令は、オペランドとして、新たなタスクの処理内容を示す命令列の先頭アドレス（以降、単に命令アドレスという）と新たなタスクのパケット５０のアドレス（以降、単にパケットアドレスという）とをとる。

【００６８】

ステップ（ａ）：プロセッサ３０は、「空き状態」のプロセッサが存在するか否かをプロセッサ状態管理装置２２に問い合わせる。このような問い合わせは、例えば、プロセッサ３０がプロセッサ状態管理装置２２にリクエスト（ＲＥＱ０＝１）を送ることにより達成される。プロセッサ状態管理装置２２は、そのリクエストに応答して「空き状態」のプロセッサが存在するか否かを判定する。

【００６９】

「空き状態」のプロセッサが存在する場合には、プロセッサ状態管理装置２２は、その「空き状態」のプロセッサの識別子をプロセッサ３０に返す。「空き状態」のプロセッサの識別子は、例えば、プロセッサ３０がプロセッサ状態管理装置２２から出力されるＩＤ０の値を参照することによって得られる。「空き状態」のプロセッサが複数個存在する場合には、優先度の最も高いプロセッサの識別子が得られる。また、複数のプロセッサが同時にｆｏｒｋ命令を解釈実行する場合には、優先度の高いプロセッサから順にｆｏｒｋ命令を解釈実行する。このようにして、プロセッサ３０は、「空き状態」のプロセッサの識別子を取得する。

【００７０】

「空き状態」のプロセッサが存在しない場合には、プロセッサ状態管理装置２２は、「空き状態のプロセッサが存在しない」旨のメッセージをプロセッサ３０に返す。「空き状態のプロセッサが存在しない」旨のメッセージは、例えば、プロセッサ３０がプロセッサ

状態管理装置２２から出力されるＮＭＰ０の値を参照することによって得られる。

【００７１】

ステップ（ｂ）：「空き状態」のプロセッサが存在した場合には、プロセッサ３０は、ステップ（ｃ）～（ｅ）の処理を行う。「空き状態」のプロセッサが存在しない場合には、プロセッサ３０は、ステップ（ｆ）～（ｇ）の処理を行う。

【００７２】

ステップ（ｃ）：ここでは、「空き状態」のプロセッサは、プロセッサ３１であると仮定する。この場合、プロセッサ３０は、ｆｏｒｋ命令のオペランドとして与えられたタスクの命令アドレスとタスクのパケットアドレスとをネットワーク２１を介してプロセッサ３１に転送する。

【００７３】

ステップ（ｄ）：プロセッサ３０は、ｆｏｒｋ命令のオペランドとして与えられたタスクのパケットアドレスによって指定されるパケット５０のロックビット領域５１に”１”を書き込み、リターンビット領域５２に”０”を書き込む。その後、プロセッサ３０は、ｆｏｒｋ命令の処理を完了し、次の命令の処理を行う。

【００７４】

ステップ（ｅ）：プロセッサ３１は、ネットワーク２１を介してプロセッサ３０からタスクの命令アドレスとタスクのパケットアドレスとを受け取る。プロセッサ３１は、受け取ったパケットアドレスによって指定されるパケット５０を参照しながら、受け取った命令アドレスによって指定される命令から処理を開始する。

【００７５】

以上のステップ（ａ）～（ｅ）により、プロセッサ３０は、プロセッサ３１によって実行される処理とは異なる処理を独立に実行することとなる。すなわち、プロセッサ３０とプロセッサ３１とによって並列処理が開始される。ｆｏｒｋ命令の処理はここで終了する。

【００７６】

ステップ（ｆ）：プロセッサ３０は、ｆｏｒｋ命令のオペランドとして与えられたタスクのパケットアドレスによって指定されるパケット５０のロックビット領域５１に”１”を書き込み、リターンビット領域５２に”１”を書き込む。また、ｆｏｒｋ命令の次の命令のアドレスをリターンアドレス領域５３に書き込む。プロセッサ３０は、実行中のタスクを中断する。

【００７７】

ステップ（ｇ）：プロセッサ３０は、ｆｏｒｋ命令のオペランドとして与えられたタスクのパケットアドレスによって指定されるパケット５０を参照しながら、ｆｏｒｋ命令のオペランドとして与えられたタスクの命令アドレスによって指定される命令から処理を開始する。ｆｏｒｋ命令の処理はここで終了する。

【００７８】

以下、図８を参照して、プロセッサ３０がｕｎｌｏｃｋ命令を解釈実行する手順をステップごとに詳細に説明する。他のプロセッサ３１および３２がｕｎｌｏｃｋ命令を解釈実行する場合も同様である。

【００７９】

ステップ（ｈ）：プロセッサ３０は、実行中のタスクが所有するパケット５０のリターンビット領域５２の値が”０”であるか否かを判定する。リターンビット領域５２の値が”０”であることは、プロセッサ３０が処理を中断したタスクが存在しないことを示す。従って、リターンビット領域５２の値が”０”である場合には、プロセッサ３０は、ステップ（ｉ）の処理を行う。リターンビット領域５２の値が”１”であることは、プロセッサ３０が処理を中断したタスクが存在することを示す。従って、リターンビット領域５２の値が”１”である場合には、プロセッサ３０は、ステップ（ｊ）の処理を行う。

【００８０】

ステップ（ｉ）：プロセッサ３０は、実行中のタスクが所有するパケット５０のロック

ビット領域５１に”０”を書き込み、プロセッサ３０の状態を「空き状態」にする。「空き状態」となったプロセッサ３０は、これ以降の処理を行わない。ｕｎｌｏｃｋ命令の処理はここで終了する。

【００８１】

ステップ（ｊ）：プロセッサ３０は、実行中のタスクが所有するパケット５０のロックビット領域５１に”０”を書き込む。さらに、プロセッサ３０は、リターンアドレス領域５３に格納されているアドレスからの命令を処理することにより、中断されたタスクを復帰させる。ｕｎｌｏｃｋ命令の処理はここで終了する。

【００８２】

表２は、ｆｏｒｋ命令およびｕｎｌｏｃｋ命令の解釈実行に応答して、マルチプロセッサシステムの状態がどのように遷移するかを示す。表２に示される例では、マルチプロセッサシステムは、プロセッサＰ１とプロセッサＰ２とを有していると仮定する。

【００８３】

【表２】

イベント	現在の状態				次の状態			
	P1	P2	T1	T2	P1	P2	T1	T2
P1.fork	RUN T1	IDLE	EX1	STOP	RUN T1	RUN T2	EX1	EX1
P2.unlock	RUN T1	RUN T2	EX1	EX1	RUN T1	IDLE	EX1	STOP
P1.fork	RUN T1	RUN other	EX1	STOP	RUN T2	RUN other	EX1	EX2
P1.unlock	RUN T2	RUN other	EX1	EX2	RUN T1	RUN other	EX1	STOP

図９に示されるように、マルチプロセッサシステムの状態は、プロセッサの状態とタスクの状態とに区分される。

【００８４】

プロセッサは、２つの状態を有する。一方の状態は「空き状態（ＩＤＬＥ）」であり、他方の状態は「実行状態（ＲＵＮ）」である。これらの状態は、プロセッサ状態管理装置２２によって管理されている状態と同じものである。プロセッサの状態が「実行状態（ＲＵＮ）」である場合には、そのプロセッサはいずれかのタスクを実行中である。

【００８５】

タスクは、３つの状態を有する。１つ目の状態は「停止状態（ＳＴＯＰ）」であり、２つ目の状態は「第１実行状態（ＥＸ１）」であり、３つ目の状態は「第２実行状態（ＥＸ２）」である。「停止状態（ＳＴＯＰ）」は、プロセッサがタスクの実行を待っている状態であるかタスクの実行を終了した状態である。「第１実行状態（ＥＸ１）」は、他のタスクの実行を中断することなく現在のタスクが実行されている状態である。「第２実行状態（ＥＸ２）」は、他のタスクの実行を中断してその後現在のタスクが実行されている状態である。プロセッサの状態が「実行状態（ＲＵＮ）」である場合には、そのプロセッサに実行されているタスクの状態は、「第１実行状態（ＥＸ１）」と「第２実行状態（ＥＸ２）」のうちのいずれかである。

【００８６】

表２を再び参照して、マルチプロセッサシステムの状態がどのように遷移するかを説明する。マルチプロセッサシステムの状態は、イベントの発生に応答して、そのイベントと現在の状態に基づいて次の状態に遷移する。ここで、「Ｐｘ．ｆｏｒｋ」という表記は、

「プロセッサＰｘがｆｏｒｋ命令を実行した」というイベントが発生したことを表し、「Ｐｘ．ｕｎｌｏｃｋ」という表記は、「プロセッサＰｘがｕｎｌｏｃｋ命令を実行した」というイベントが発生したことを表す。

【００８７】

表２の第１行は、プロセッサＰ１が「実行状態」（タスクＴ１を実行中）であり、プロセッサＰ２が「空き状態」であり、タスクＴ１が「第１実行状態」であり、タスクＴ２が「停止状態」である場合において、「プロセッサＰ１がｆｏｒｋ命令を実行した」というイベントに応答して、プロセッサＰ２の状態が「空き状態」から「実行状態」（タスクＴ２を実行中）に変更され、タスクＴ２の状態が「停止状態」から「第１実行状態」に変更されることを示す。このように状態が遷移するのは、新たなタスクＴ２が生成された時点でタスクＴ２が「空き状態」のプロセッサＰ２に割り当てられるからである。

【００８８】

表２の第２行は、表２の第１行における次の状態が現在の状態である場合において、「プロセッサＰ２がｕｎｌｏｃｋ命令を実行した」というイベントに応答して、プロセッサＰ２の状態が「実行状態」（タスクＴ２を実行中）から「空き状態」に変更され、タスクＴ２の状態が「第１実行状態」から「停止状態」に変更されることを示す。

【００８９】

表２の第３行は、プロセッサＰ１が「実行状態」（タスクＴ１を実行中）であり、プロセッサＰ２が「実行状態」（他のタスクを実行中）であり、タスクＴ１が「第１実行状態」であり、タスクＴ２が「停止状態」である場合において、「プロセッサＰ１がｆｏｒｋ命令を実行した」というイベントに応答して、プロセッサＰ１の状態が「実行状態」（タスクＴ１を実行中）から「実行状態」（タスクＴ２を実行中）に変更され、タスクＴ２の状態が「停止状態」から「第２実行状態」に変更されることを示す。このように状態が遷移するのは、新たなタスクＴ２が生成された時点で「空き状態」のプロセッサが存在しないため、プロセッサＰ１がタスクＴ１の実行を中断してタスクＴ２の実行を開始するからである。

【００９０】

表２の第４行は、表２の第３行における次の状態が現在の状態である場合において、「プロセッサＰ１がｕｎｌｏｃｋ命令を実行した」というイベントに応答して、プロセッサＰ１の状態が「実行状態」（タスクＴ２を実行中）から「実行状態」（タスクＴ１を実行中）に変更され、タスクＴ２の状態が「第２実行状態」から「停止状態」に変更されることを示す。

【００９１】

以下、ｆｏｒｋ命令とｕｎｌｏｃｋ命令を含むプログラムを並列処理する場合におけるマルチプロセッサシステム１の動作を説明する。

【００９２】

図１０は、１から４までの和（１＋２＋３＋４）を二分木に基づいて計算するプログラムの手順を示す。このプログラムは、ｍａｉｎとｓｕｍの２つの部分に分かれており、ｍａｉｎは主プログラム、ｓｕｍは再帰呼び出し可能でかつ並列処理可能なサブルーチンである。ｓｕｍはｎとｍの２つの引数をとり、ｎ＋１からｍまでの和を求めるものである。ｍａｉｎはｎ＝０、ｍ＝４を引数としてｓｕｍを呼び出すものである。

【００９３】

まず、初期状態として、プロセッサ３０はｍａｉｎを実行していると仮定する。プロセッサ３０の状態は「実行状態」である。また、プロセッサ３１およびプロセッサ３２の状態は「空き状態」であると仮定する。

【００９４】

以下、プログラムの各ステップ（Ａ）～（Ｈ）について、マルチプロセッサシステム１がどのように動作するかを詳細に説明する。

【００９５】

ステップ（Ａ）：プロセッサ３０は、ｎ＝０、ｍ＝４を引数としてｓｕｍサブルーチン

を実行する。具体的には、プロセッサ３０は、共有キャッシュメモリ２０上にパケット５０（Ｐｋ１）を確保し、そのパケット５０（Ｐｋ１）の引数領域５４に値０と値４とを格納する。次に、プロセッサ３０は、ｓｕｍの命令の先頭アドレスとパケット５０（Ｐｋ１）の先頭アドレスとをオペランドとして、ｅｘｅｃ命令を実行する。ｅｘｅｃ命令とは、図７に示すｆｏｒｋ命令の処理手順のうちステップ（ｆ）と（ｇ）のみに対応する命令である。ｅｘｅｃ命令は、ｆｏｒｋ命令と同様にして、オペランドとしてタスクの命令アドレスとタスクのパケットアドレスとをとる。

【００９６】

プロセッサ３０は、パケット５０（Ｐｋ１）のロックビット領域５１に”１”を書き込み、パケット５０（Ｐｋ１）のリターンビット領域５２に”１”を書き込み、リターンアドレス領域５３にｅｘｅｃ命令の次の命令のアドレスを格納する（図７のステップ（ｆ）を参照）。また、プロセッサ３０は、パケット５０（Ｐｋ１）を参照しながらｓｕｍの命令の実行を開始する（図７のステップ（ｇ）を参照）。

【００９７】

ステップ（Ｂ）：プロセッサ３０は、パケット５０（Ｐｋ１）から引数ｎと引数ｍとを読み出し、（ｎ＋１）とｍとを比較する。（ｎ＋１）とｍが等しい場合には、処理はステップ（Ｇ）に進み、その他の場合には、処理はステップ（Ｃ）に進む。ｓｕｍサブルーチンがｍａｉｎから最初に呼ばれた場合には、ｎ＝０、ｍ＝４であるから、（ｎ＋１）とｍとは等しくない。従って、処理は、ステップ（Ｃ）に進む。

【００９８】

ステップ（Ｃ）：プロセッサ３０は、ｋ＝（ｎ＋ｍ）ｄｉｖ２を計算する。ここで、（ｎ＋ｍ）＝４であるから、ｋ＝２となる。

【００９９】

ステップ（Ｄ）：プロセッサ３０は、ｎとｋとを引数としてｓｕｍサブルーチンを実行する。具体的には、プロセッサ３０は、共有キャッシュメモリ２０上にパケット５０（Ｐｋ２）を確保し、そのパケット５０（Ｐｋ２）の引数領域５４に値ｎ（＝０）と値ｋ（＝２）とを格納する。次に、プロセッサ３０は、ｓｕｍの命令の先頭アドレスとパケット５０（Ｐｋ２）の先頭アドレスとをオペランドとして、ｆｏｒｋ命令を実行する。

【０１００】

プロセッサ３１とプロセッサ３２はいずれも「空き状態」である。プロセッサ３０は、優先度に従って「空き状態」のプロセッサ３１の識別子を得る（図７のステップ（ａ）を参照）。プロセッサ３０は、タスクの命令アドレスとタスクのパケットアドレスとをプロセッサ３１に転送する（図７のステップ（ｂ）を参照）。プロセッサ３０は、パケット５０（Ｐｋ２）のロックビット領域５１に”１”を書き込み、パケット５０（Ｐｋ２）のリターンビット領域５２に”０”を書き込む（図７のステップ（ｄ）を参照）。さらに、プロセッサ３１は、パケット５０（Ｐｋ２）を参照しながらｓｕｍの命令の実行を開始する（図７のステップ（ｅ）を参照）。このようにして、プロセッサ３０とプロセッサ３１とはｓｕｍサブルーチンを並列に実行する。

【０１０１】

ステップ（Ｅ）：プロセッサ３０は、ｋとｍとを引数としてｓｕｍサブルーチンを実行する。具体的には、プロセッサ３０は、共有キャッシュメモリ２０上にパケット５０（Ｐｋ３）を確保し、そのパケット５０（Ｐｋ３）の引数領域５４に値ｋ（＝２）と値ｍ（＝４）とを格納する。次に、プロセッサ３０は、ｓｕｍの命令の先頭アドレスとパケット５０（Ｐｋ３）の先頭アドレスとをオペランドとして、ｅｘｅｃ命令を実行する。プロセッサ３０がｅｘｅｃ命令の実行を開始する前に、パケット５０（Ｐｋ１）はスタック領域に退避される。

【０１０２】

プロセッサ３０は、パケット５０（Ｐｋ３）のロックビット領域５１に”１”を書き込み、パケット５０（Ｐｋ３）のリターンビット領域５２に”１”を書き込み、リターンアドレス領域５３にｅｘｅｃ命令の次の命令のアドレスを格納する（図７のステップ（ｆ）

を参照）。また、プロセッサ３０は、パケット５０（Ｐｋ３）を参照しながらｓｕｍの命令の実行を開始する（図７のステップ（ｇ）を参照）。

【０１０３】

ステップ（Ｆ）：プロセッサ３０は、ステップ（Ｅ）において呼び出したｓｕｍサブルーチンの実行を終了した後、スタック領域に退避したパケット５０（Ｐｋ１）を復帰させる。その後、プロセッサ３０は、ｓ１とｓ２とを加算する。ここで、ｓ１は、ステップ（Ｄ）において実行されたｓｕｍサブルーチンの結果を示す。従って、ｓ１は、パケット５０（Ｐｋ２）の戻り値領域５５に格納される。ｓ２は、ステップ（Ｅ）において実行されたｓｕｍサブルーチンの結果を示す。従って、ｓ２は、パケット５０（Ｐｋ３）の戻り値領域５５に格納される。プロセッサ３０がステップ（Ｅ）において呼び出したｓｕｍサブルーチンの実行を終了した時点では、パケット５０（Ｐｋ２）を所有するタスクはまだ実行中である可能性がある。プロセッサ３０は、パケット５０（Ｐｋ２）を所有するタスクの実行が終了した後、パケット５０（Ｐｋ２）の戻り値領域５５に格納されている値を読み出し、その値をｓ１とする。ここでは、ｓ１＝３である。パケット５０（Ｐｋ２）を所有するタスクの実行が終了したか否かは、パケット５０（Ｐｋ２）のロックビット領域５１の値を参照することにより判定される。パケット５０（Ｐｋ２）のロックビット領域５１の値が”０”であることは、パケット５０（Ｐｋ２）を所有するタスクの実行が終了したことを示す。

【０１０４】

同様にして、プロセッサ３０は、パケット５０（Ｐｋ３）を所有するタスクの実行が終了した後、パケット５０（Ｐｋ３）の戻り値領域５５に格納されている値を読み出し、その値をｓ２とする。ここでは、ｓ２＝７である。プロセッサ３０は、ｓ１＋ｓ２を計算する。その結果、ｓ＝１０が得られる。

【０１０５】

ステップ（Ｈ）：プロセッサ３０は、ｓの値をパケット５０（Ｐｋ１）の戻り値領域５５に格納する。その後、プロセッサ３０は、ｕｎｌｏｃｋ命令を実行する。

【０１０６】

プロセッサ３０は、パケット５０（Ｐｋ１）のリターンビット領域５２に格納されている値が”１”であるか否かを判定する（図８のステップ（ｈ）を参照）。は、”１”である。従って、プロセッサ３０は、パケット５０（Ｐｋ１）のロックビット領域５１に”０”を格納し、リターンアドレス領域５３に格納されているアドレスからの命令を実行する（図８のステップ（ｊ）を参照）。この場合、ｍａｉｎのステップ（Ａ）の次の命令から処理が再開される。

【０１０７】

ステップ（Ｇ）：ステップ（Ｂ）において、ｎ＋１＝ｍであると判定された場合は、処理はステップ（Ｇ）に進む。プロセッサ３０は、ｓに引数ｍの値を代入する。その後、処理はステップ（Ｈ）に進む。

【０１０８】

ここで、ステップ（Ｄ）において呼び出されたｓｕｍサブルーチンやステップ（Ｅ）において呼び出されたｓｕｍサブルーチンにおいても、上述したステップ（Ｂ）～（Ｈ）が実行されることに注意されたい。ｓｕｍサブルーチンは、再帰呼び出し可能なサブルーチンだからである。

【０１０９】

このように、ｓｕｍサブルーチンを再帰的に呼び出すことにより、１から４の和（１＋２＋３＋４）を並列に計算することが達成される。この例では、ステップ（Ｄ）におけるｆｏｒｋ命令とステップ（Ｅ）におけるｅｘｅｃ命令によって２つのタスクが生成されている。ｆｏｒｋ命令は「空き状態」のプロセッサがある限りそのプロセッサにタスクを割り当てるために使用される命令であり、ｅｘｅｃ命令は、あくまで自プロセッサにタスクを割り当てるために使用される命令である。

【０１１０】

図１１は、上述した処理の内容を模式的に示したものである。図１１に示されるように、タスクｓｕｍ（０，４）からｆｏｒｋ命令とｅｘｅｃ命令とにより２つのタスクｓｕｍ（０，２）とタスクｓｕｍ（２，４）とが生成される。タスクｓｕｍ（０，２）はプロセッサ３１に割り当てられ、タスクｓｕｍ（２，４）はプロセッサ３０に割り当てられる。同様に、２つのタスクのそれぞれからさらに２つのタスクが生成される。「空き状態」のプロセッサが存在する限り他のプロセッサにタスクが割り当てられる。

【０１１１】

タスクｓｕｍ（２，４）からタスクｓｕｍ（２，３）とタスクｓｕｍ（３，４）とが生成される。しかし、いずれのタスクもプロセッサ３０に割り当てられる。タスク（２，３）の割り当て時に「空き状態」のプロセッサがすでに存在しなくなっているからである。

【０１１２】

このように、本発明のマルチプロセッサシステム１におけるプロセッサ３０～３２のそれぞれは、ｆｏｒｋ命令を解釈実行することにより、「空き状態」のプロセッサが存在する場合にはそのプロセッサにタスクを割り当て、「空き状態」のプロセッサが存在しない場合には実行中のタスクの実行を中断して、そのプロセッサにタスクを割り当てる。このようにして、処理すべきタスクが生成されると同時に「空き状態」のプロセッサか、あるいはタスクを生成したプロセッサのいずれかにその生成されたタスクが割り当てられる。その結果、生成さたタスクは即時に実行される。これにより、従来のマルチプロセッサシステムでは必要とされた処理すべきタスクを保存する機構や、タスクの実行順序をスケジューリングする機構は不要となる。また、「空き状態」のプロセッサが存在する場合には、必ずそのプロセッサにタスクが割り当てられるため、プロセッサの利用効率も高い。

【０１１３】

さらに、ｆｏｒｋ命令やｕｎｌｏｃｋ命令は簡単なハードウェアで実現することができ、高速な処理も実現することができる。

【０１１４】

従って、集積回路上に実装されたマルチプロセッサシステム１において、例示した０から４までの和を求めるプログラムのような、タスクの処理時間がスケジューリング処理時間や実行待ちタスクの管理処理に要する時間に比べて小さいプログラムを並列処理する場合には、本発明のタスク実行方法は非常に有用である。

【０１１５】

なお、集積回路の外部から割り込みが入った場合には、プロセッサ状態管理装置２２を用いて「空き状態」のプロセッサを検出し、「空き状態」のプロセッサのうち最も優先度の低いプロセッサに割り込み処理を行わせることにより、割り込み処理による性能低下を低減できる。

【０１１６】

なお、集積回路のプロセッサがすべて「空き状態」になったことは、プロセッサ状態管理装置２２を用いて検出することができる。従って、この場合には、いずれかのプロセッサで例外処理を行うことによりデッドロックを回避することができる。

【０１１７】

以上のように、本発明によれば、あるプロセッサで新たなタスクを生成したときにそのタスクの実行を他あるいは自プロセッサによりただちに開始することができる。このことは、タスクを保持しておく機構やタスクの実行順序をスケジューリングする機構を不要にする。また、実行待ちのタスクを選択し、その選択されたタスクを「空き状態」のプロセッサに割り当てる処理も不要となる。

【０１１８】

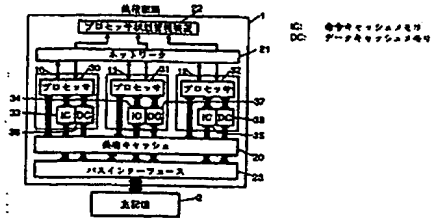
その結果、タスクの処理時間に比較してプロセッサ割り当てに要する時間が少なくてすむ。これにより、マルチプロセッサシステムにおいて、粒度の細かい並列処理の高速化を図ることができる。

【図面の簡単な説明】

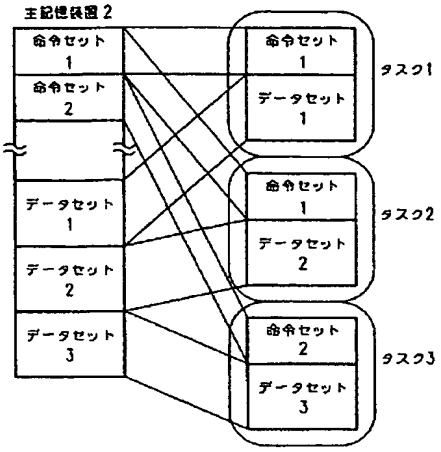
【０１１９】

【図１】本発明のマルチプロセッサシステム１の構成を示す図である。
【図２】タスクの概念を模式的に示す図である。
【図３】マルチプロセッサシステム１におけるプロセッサ状態管理装置２２の構成例を示す図である。
【図４】（ａ）および（ｂ）は、プロセッサ状態管理装置２２の動作の一例を説明する図である。
【図５】（ａ）および（ｂ）は、プロセッサ状態管理装置２２の動作の他の一例を説明する図である。
【図６】パケット５０の構成を示す図である。
【図７】プロセッサ３０～３２がｆｏｒｋ命令を解釈実行する手順を示す図である。
【図８】プロセッサ３０～３２がｕｎｌｏｃｋ命令を解釈実行する手順を示す図である。
【図９】プロセッサの状態とタスクの状態とを説明する図である。
【図１０】１から４までの和を二分木に基づいて計算するプログラムの手順を示す図である。
【図１１】図１０に示すプログラムの処理の内容を模式的に示した図である。
【図１２】従来のプロセッサ割当方法の動作を説明する図である。
【図１３】タスクが中粒度～粗粒度である場合における、タスクの処理時間とオーバヘッドの処理時間とを示すタイムチャートである。
【図１４】タスクが細粒度である場合における、タスクの処理時間とオーバヘッドの処理時間とを示すタイムチャートである。
【符号の説明】
【０１２０】
１　マルチプロセッサシステム
２　主記憶装置
１０～１２　要素プロセッサユニット
２０　共有キャッシュ
２１　ネットワーク
２２　プロセッサ状態管理装置
２３　バスインターフェース
３０～３２　プロセッサ
３３～３５　命令キャッシュ（ＩＣ）
３６～３８　データキャッシュ（ＤＣ）

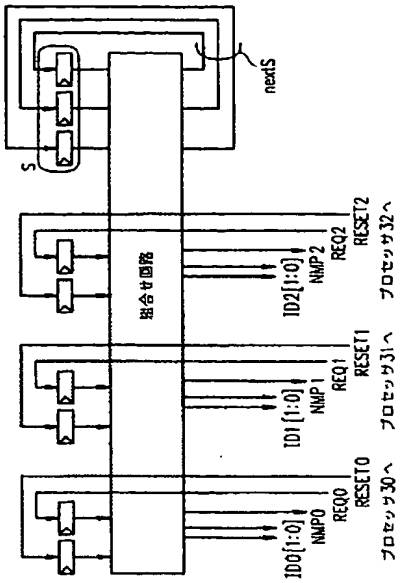
【図１】



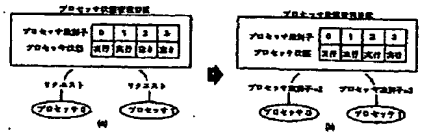
【図２】



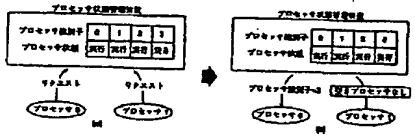
【図３】



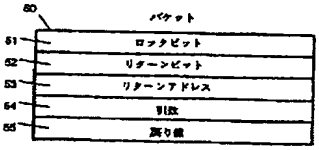
【図４】



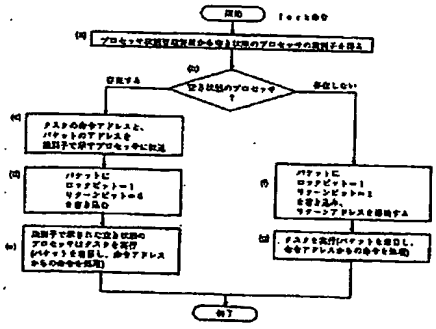
【図５】



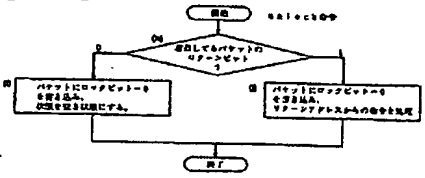
【図６】



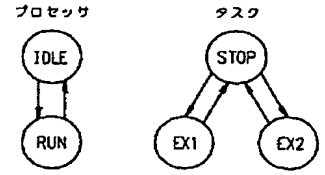
【図７】



【図８】



【図９】

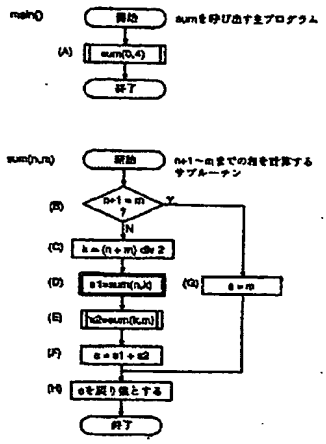


IDLE アイドル　STOP 実行待ちまたは実行済

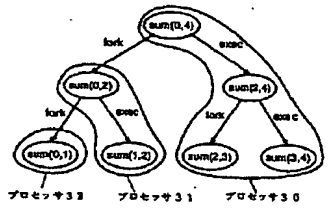
RUN 実行中　EX1 実行中(他タスク中断なし)

EX2 実行中(他タスク中断あり)

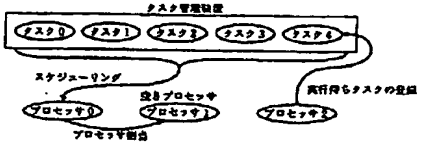
【図１０】



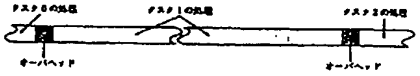
【図１１】



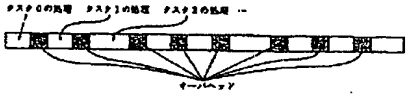
【図１２】



【図１３】



【図１４】



フロントページの続き

(72)発明者 田沼 仁
大阪府三島郡島本町広瀬４丁目１１番６号

Ｆターム(参考) 5B045 DD01 DD12 GG03 GG11
5B098 AA10 GA04 GA07 GA08